PSTAT231 Hw5 muxi

muxi

2022-11-20

Exercise 1

```
library(tidymodels)
```

```
## ── Attaching packages ─────────────────────────────────────── tidymodels 1.0.0 ──
```

```
## ✓ broom        1.0.1      ✓ recipes      1.0.2
## ✓ dials        1.0.0      ✓ rsample      1.1.0
## ✓ dplyr        1.0.10     ✓ tibble       3.1.8
## ✓ ggplot2      3.3.6      ✓ tidyr        1.2.1
## ✓ infer        1.0.3      ✓ tune         1.0.1
## ✓ modeldata    1.0.1      ✓ workflows    1.1.0
## ✓ parsnip      1.0.2      ✓ workflowsets 1.0.0
## ✓ purrr        0.3.4      ✓ yardstick    1.1.0
```

```
## ── Conflicts ────────────────────────────────────────── tidymodels_conflicts() ──
## ✖ purrr::discard() masks scales::discard()
## ✖ dplyr::filter()  masks stats::filter()
## ✖ dplyr::lag()     masks stats::lag()
## ✖ recipes::step()  masks stats::step()
## • Search for functions across packages at https://www.tidymodels.org/find/
```

```
library(ISLR)
library(ISLR2)
```

```
##
## 载入程辑包：'ISLR2'
```

```
## The following objects are masked from 'package:ISLR':
##
##     Auto, Credit
```

```
library(tidyverse)
```

```
## ── Attaching packages
## ─────────────────────────────────────── tidyverse 1.3.2 ──
```

```
## ✓ readr   2.1.3     ✓ forcats 0.5.2
## ✓ stringr 1.4.1
## ── Conflicts ──────────────────────────────────────── tidyverse_conflicts() ──
## ✖ readr::col_factor() masks scales::col_factor()
## ✖ purrr::discard()    masks scales::discard()
## ✖ dplyr::filter()     masks stats::filter()
## ✖ stringr::fixed()    masks recipes::fixed()
## ✖ dplyr::lag()        masks stats::lag()
## ✖ readr::spec()       masks yardstick::spec()
```

```
tidymodels_prefer()
library(janitor)
library(pROC)
```

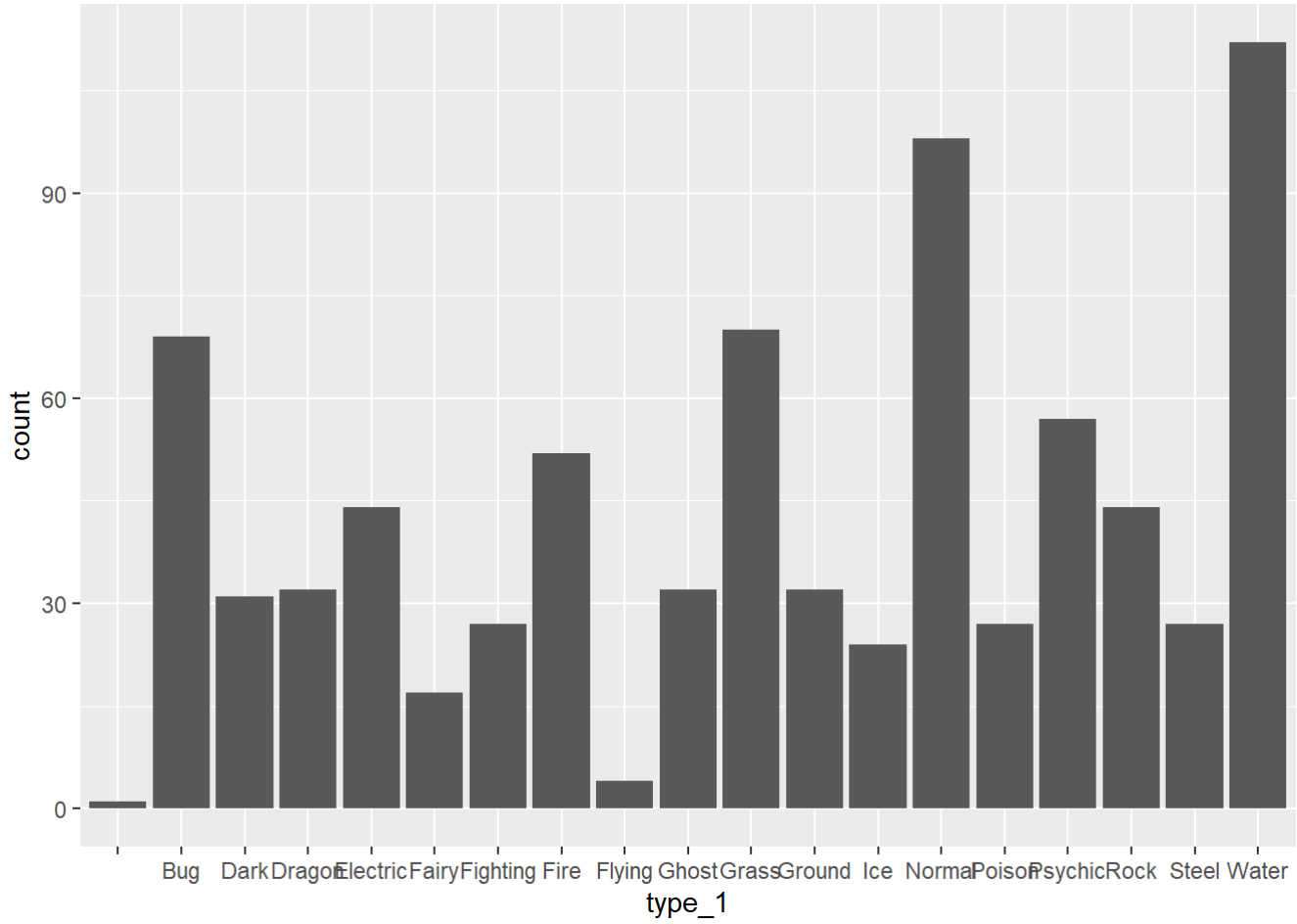
```
## Type 'citation("pROC")' for a citation.
```

```
data=read.csv("Pokemon.csv")%>%clean_names()
```

Resulting names of the dataframe are unique and consist only of the _ character, numbers, and letters. Returns the data.frame with clean names.

Exercise 2

```
data %>%
  ggplot(aes(x = type_1)) +
  geom_bar()
```



We could see from the bar chart that there are total 19 classes of the outcome. And there one Pokémon type–Flying with very few Pokémon.

```
data=data %>% filter(type_1=="Bug"|type_1=="Fire"|type_1=="Grass"|type_1=="Normal"|type_1=="Water"|type_1=="Psychic")
data[,3]=as.factor(data[,3])
data[,13]=as.factor(data[,13])
class(data[,3])
```

```
## [1] "factor"
```

```
class(data[,13])
```

```
## [1] "factor"
```

Exercise 3

```
set.seed(1215)
data_split=initial_split(data, strata = type_1, prop = 0.7)
data_train=training(data_split)
data_test=testing(data_split)
dim(data_train)
```

```
## [1] 318  13
```

```
dim(data_test)
```

```
## [1] 140  13
```

```
data_folds=vfold_cv(data_train, v = 5,strata = type_1)
data_folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 × 2
##   splits           id
##   <list>           <chr>
## 1 <split [252/66]> Fold1
## 2 <split [253/65]> Fold2
## 3 <split [253/65]> Fold3
## 4 <split [256/62]> Fold4
## 5 <split [258/60]> Fold5
```

Using strata, we could set a variable in data used to conduct stratified sampling. Using k-Fold Cross-Validation could help us improve the accuracy for models with hyperparameter tuning.

Exercise 4

```
ENT_recipe=recipe(type_1 ~ legendary+generation+sp_atk+attack+speed+defense+hp+sp_def, data = data_train) %>% step_dummy(legendary,generation)%>%step_center(all_predictors())%>%step_scale(all_predictors())
```

Exercise 5

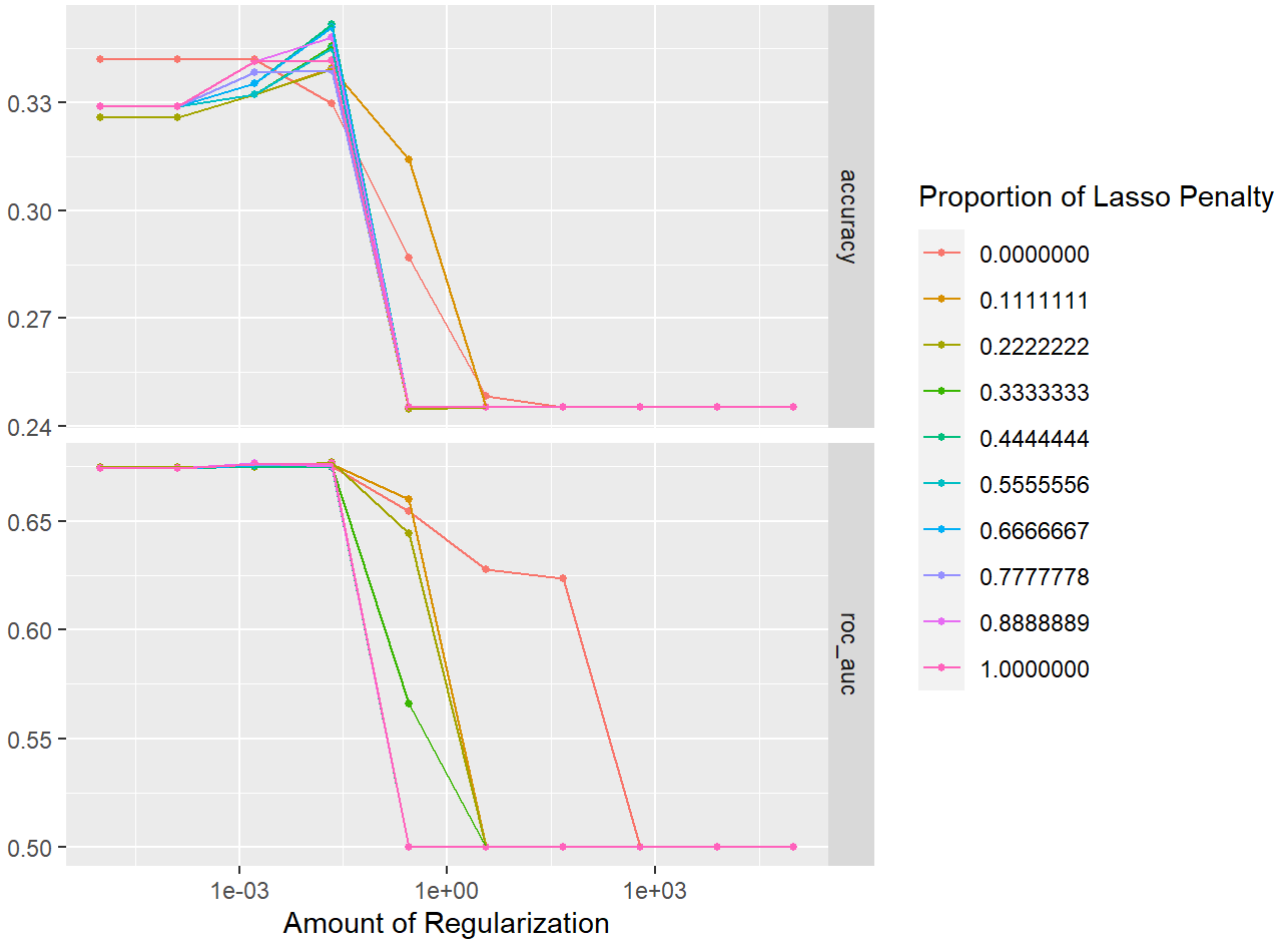
```
Elastic_Net_Tuning=multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

ENT_workflow=workflow() %>%
  add_recipe(ENT_recipe) %>%
  add_model(Elastic_Net_Tuning)

para_grid=grid_regular(penalty(range = c(-5, 5)),mixture(range = c(0, 1)), levels = c(mixture = 10, penalty = 10))
```

There are totally 100 models.

Exercise 6

```
ENT_res=tune_grid(
  ENT_workflow,
  resamples = data_folds,
  grid = para_grid
)
```

```
autoplot(ENT_res)
```



It is clear that when the amount of regularization are extremly large, both accuracy and roc auc shows a poor curve. For accuracy plot, the curves first increase with the amount of regularization to the highest point and then keep decreasing. For roc auc plot, the curves keep stable for a period and then keep decreasing.

For accuracy plot, the curves are not monotone. We could not conclude that larger or smaller values of penalty and mixture produce better accuracy.

For roc auc plot, smaller values of penalty and mixture produce better ROC AUC.

Exercise 7

```
best_para=select_best(ENT_res, metric = "roc_auc")
best_para
```

```
## # A tibble: 1 × 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1  0.0215   0.222 Preprocessor1_Model024
```

```
ENT_final=finalize_workflow(ENT_workflow, best_para)
ENT_final_fit=fit(ENT_final, data = data_train)
ENT_acc=augment(ENT_final_fit, new_data = data_test) %>%accuracy(truth = type_1, estimate = .
pred_class)
ENT_acc
```

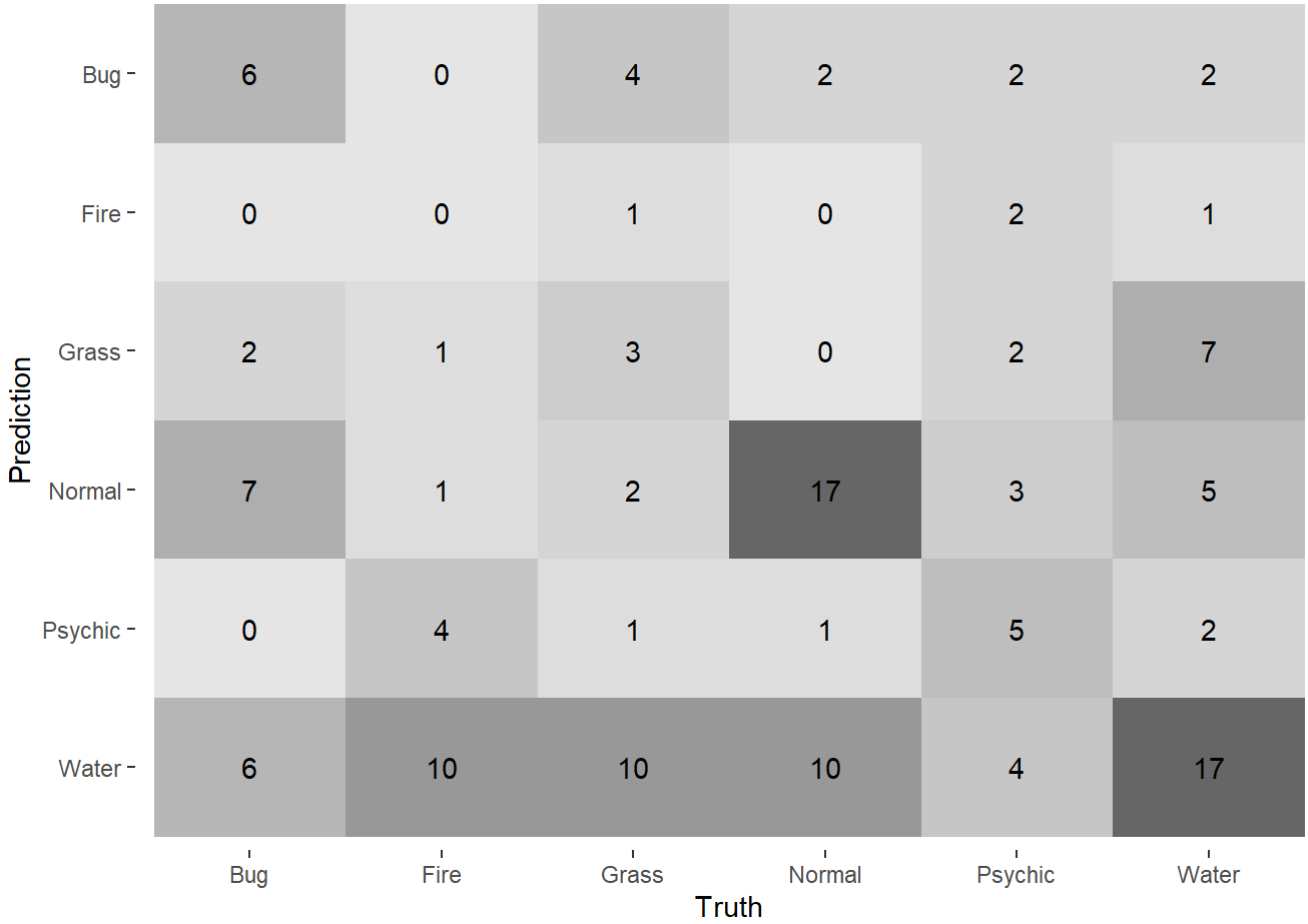
```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.343
```

Exercise 8

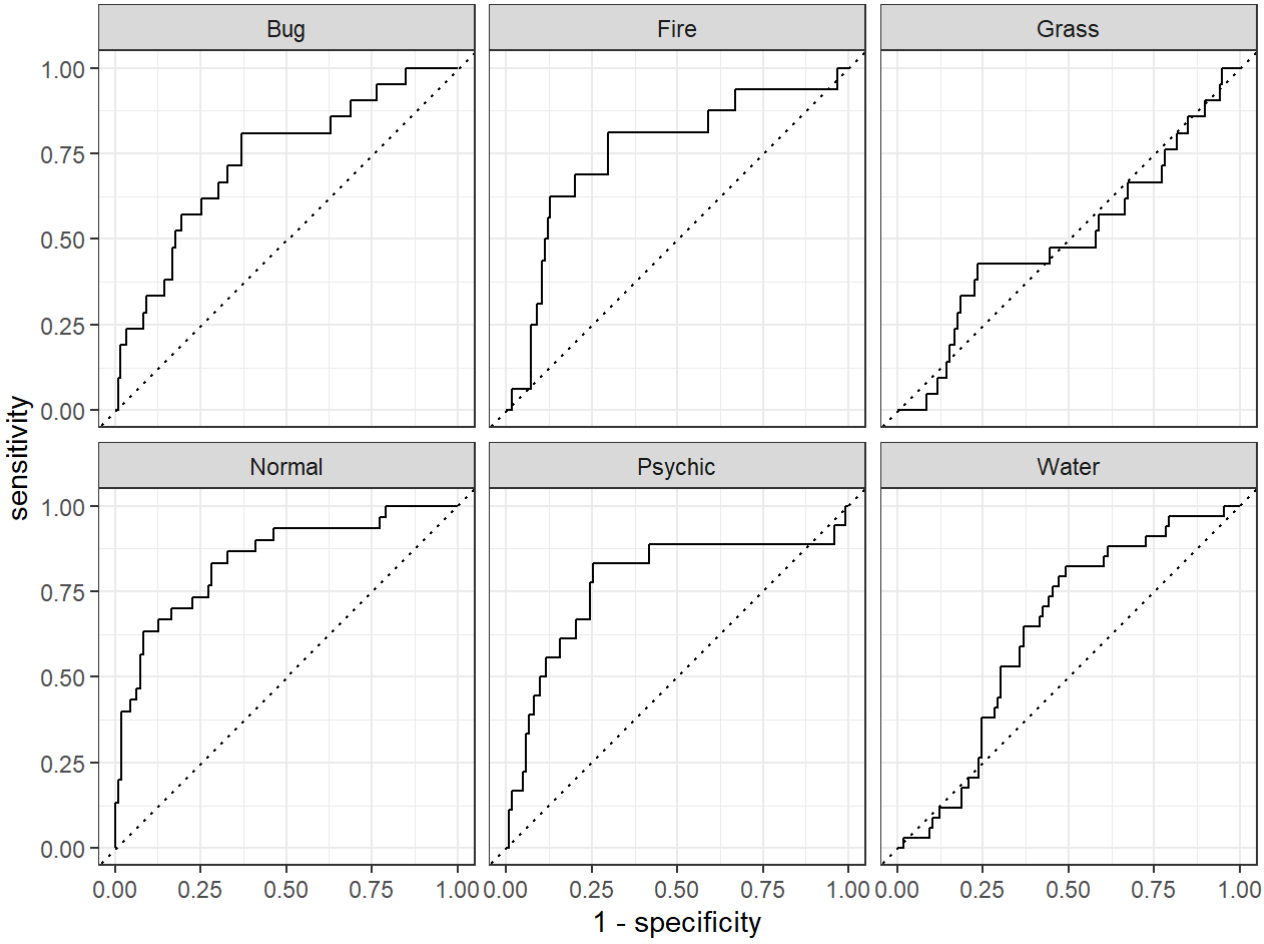
```
k1=predict(ENT_final_fit, data_test, type="prob")
k2=predict(ENT_final_fit, data_test, type="class")
k3=cbind(k1,k2)
Pre=cbind(data_test[,3],k3)
Pre %>%roc_auc(data_test[,3],  .pred_Bug: .pred_Water)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.706
```

```
augment(ENT_final_fit, new_data = data_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```




```
Pre %>%
  roc_curve(data_test[,3],  .pred_Bug: .pred_Water) %>%
  autoplot()
```

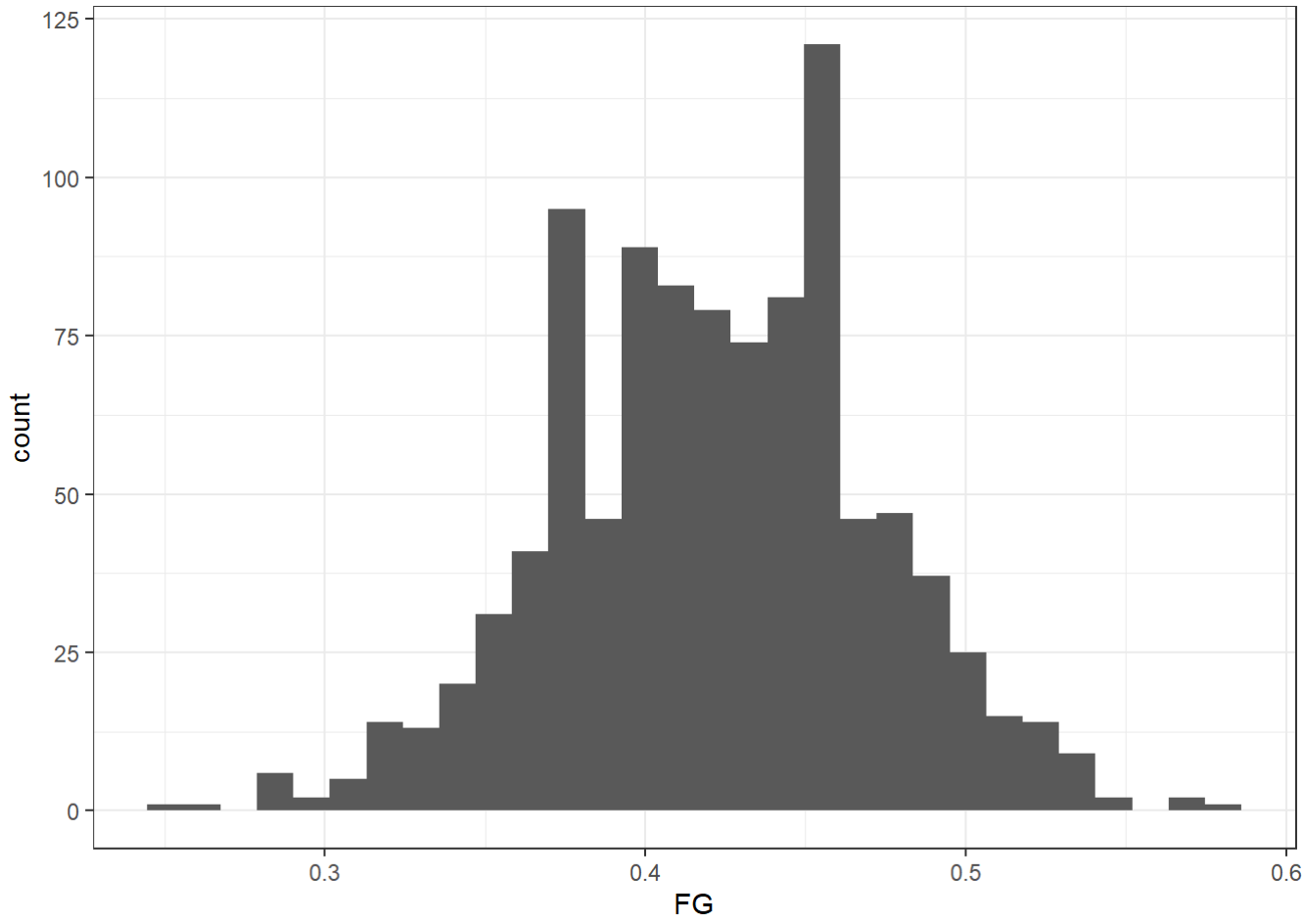


From the plots, we could see that the predicting accuracy is not the same for the six kind of Pokemon. Some perform good and some perform bad. The model is best at predicting the normal type Pokemon and worst at the Fire type Pokemon. In my opinion, the significant characteristic of the Fire type Pokemon may not be collected. Thus, we could not catch the key features of them.

Exercise 9

```
X=c(numeric(337)+1,numeric(464))
FG=numeric(1000)
i=1
while(i<1001){
s=sample(X,100, replace = TRUE)
FG[i]=mean(s)
i=i+1
}
fg=as.data.frame(FG)
fg %>%
  ggplot(aes(x = FG)) +
  geom_histogram() +
  theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
quantile(FG, probs = c(0.005, 0.995))
```

```
##    0.5%   99.5%
## 0.28995 0.54005
```

```
print(quantile(FG, probs = c (0.995)))
```

```
##   99.5%
## 0.54005
```